CHOMP: A Framework and Instruction Set for Latency Tolerant, Massively Multithreaded Processors

John Leidel, Kevin Wadleigh, Joe Bolding, Tony Brewer, Dean Walker

IA^3 Workshop on Irregular Applications: Architectures and Algorithms
Overview

- Motivations
- MX-100 Hardware Overview
- CHOMP Personality Overview
- CHOMP Instruction Set
- Application Examples
An HPC Programmer’s Wish List

• Globally shared, single word memory access
• Lightweight synchronization and barrier operators
• Latency hiding techniques
• Simple parallel scalability
• Bandwidth!
• ...all with familiar and/or commodity programming models
  – Not all programming models are created equal
  – None are perfect, but industry adoption is paramount

IA^3 2012
Architectural Wish List

- Globally Shared Memory
- Single Word Memory Access
- Lightweight Sync/Barriers
- Latency Hiding
- Bandwidth
- Simple Scalability
- Commodity Programming Model
- Memory Interconnect
- Scatter/Gather Memory
- Atomic Memory Operations
- Tag-Bit Semantics
- Memory & Compute Overlap
- Wide, Concurrent Memory Bus
- Develop on your laptop, execute on the super
- Portable Code

IA^3 2012
MX-100 Platform

- HCMI = Hybrid Core Memory Interconnect; PCIe Gen2 X8 link
- All memory, host and coprocessor, is globally shared and virtually addressable
MX-100 Coprocessor

Shared, Virtual Memory System

- **Scatter/Gather**
  - 1TB of coprocessor memory
  - 128 GB/s bandwidth to coprocessor memory

- **Atomic Memory Operators**
  - Native to memory controllers
  - 8-Byte Accessible
    - Does not require fetching cachelines

- **Tag Bit Semantics**
  - lock or “tag” bit for each 8-byte word
MX-100 Single-node Block Diagram

Host

AEH

AE0

AE1

AE2

AE3

64MB pages

Atomic ops

TLB

snoop

aops

sched

SG-DIMM

TLB

snoop

aops

sched

SG-DIMM

TLB

snoop

aops

sched

SG-DIMM

TLB

snoop

aops

sched

SG-DIMM

TLB

snoop

aops

sched

SG-DIMM

1024 TIDs per link
(32K OR)
32TB (45 bit)
physical address space

32 DDR3 SG-DIMMs
HalfDIMM up to 4GB - 256GB system
FullDIMM up to 32GB - 1TB system

IA^3 2012
Atomic Operations & FE Memory Bits

• Atomic operations avoid round trips to memory to acquire lock, update data, release lock
  – Accessible from coprocessor instruction space
  – Accessible from host via lock engine and compiler intrinsics

• Full/Empty Bits
  – Extra bit stored with each word
  – Can be used to signify when data is valid/ready
  – Accessible from host via lock engine & compiler intrinsics

<table>
<thead>
<tr>
<th>Atomic Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add, Sub</td>
</tr>
<tr>
<td>Min, Max</td>
</tr>
<tr>
<td>Exch</td>
</tr>
<tr>
<td>Inc, Dec</td>
</tr>
<tr>
<td>CAS</td>
</tr>
<tr>
<td>And, Or, Xor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Bit Semantic Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriteEF, WriteFF</td>
</tr>
<tr>
<td>WriteXF, WriteXE</td>
</tr>
<tr>
<td>ReadFE, ReadEF, ReadFF</td>
</tr>
<tr>
<td>ReadXX</td>
</tr>
<tr>
<td>IncFF</td>
</tr>
</tbody>
</table>
CHOMP Personality Overview
CHOMP: Convey Hybrid

• **Scalable, MIMD Personality Framework**
  – Simple, RISC-like instruction set
  – Atomic memory operations
  – Fine-grained synchronization using tag-bit semantics
  – Software driven, hardware-based low-latency thread/task scheduler
  – *Every instruction is treated as a first class citizen*

• **First Programming model is OpenMP 3.0**
  – Support for vanilla OpenMP code directives
  – Simple, portable parallelism
  – Convey has joined the OpenMP Architecture Review Board

• **Interest in other language constructs [DSL’s]**
CHOMP Architecture Hierarchy

- Multi-threaded/Tasking Applications
  - Convey Compiler
    - Other High Level Runtime
    - OpenMP
      - Convey Lightweight Runtime
      - Convey Perflib
      - CHOMP Instruction Set
CHOMP Personality Infrastructure

Function Pipe Group

Dispatch Interface

Memory Crossbar

CHOMP Application Engine

Instruction Cache

Thread Cache Unit

Hardware Scheduler

Function Pipe//ALU

Function Pipe//ALU

Hardware Scheduler

Hardware Scheduler

Function Pipe//ALU

Memory Interface

Function Pipe

Application Engine

IA^3 2012
CHOMP Thread Cache Units

- **Smallest divisible unit of parallelism in hardware**
  - Control register file
  - User register file

- **A single TCU maps to some autonomous unit of software parallelism**
  - Thread, task, fiber, pebble, etc
  - In OpenMP, each TCU represents a thread

- **Scheduling decisions and context switching is performed on a TCU by TCU basis**

- **The current personalities have 64 TCU’s per Function Pipe**
  - 63 are available to the user
  - 1 is used for the Workload Manager
CHOMP Function Pipe/Function Pipe Group

- **Function Pipes**
  - Contains arithmetic unit(s)
    - First personality design will contain:
      - Integer Mul, Add, Misc
      - Double Precision Floating Point Add, Mul
    - Multiple Thread Cache Units share a Function Pipe
      - The first personality contains 64 TCU’s per FP
  - **Workload Manager**
    - Manages the scheduling on TCU’s access to Function Pipe resources

- **Function Pipe Groups**
  - 8-way Set Associative Instruction Cache
    - ICACHE shared amongst all FP’s+TCU’s
  - Contains one or more Function Pipes
  - Memory interface to AE crossbar
CHOMP Personality Infrastructure

MX-100 DP Floating Point Personality
- 4 Application Engines per Coprocessor
- 12 Function Pipes per Application Engine
- 64 Threads per Function Pipe
- 3024 total threads per Coprocessor

4 x CHOMP Application Engines per coprocessor
CHOMP TCU Scheduling

• Any time a Workload Manager finds empty TCU’s, it will attempt to fetch them from the runtime work queues

• The Workload Manager will fetch:
  – \( MIN(\ \text{Thread\_Cache\_Count}, \ \text{Empty\_TCU\_Count}) \)
  – The Thread Cache Count [TCC] values will affect the hardware’s ability to naturally balance the load
  – The default TCC value is the number of TCU’s per FP [in this case 64]
CHOMP TCU Scheduling cont.

• The hardware enforces a TDM round-robin policy on a single cycle between TCU’s
  – A minimum of 16 TCU’s per FP must be active in order to not stall the Function Pipe

• The hardware will not select TCU’s for execution that are in the following state:
  – Register hazarded
  – Waiting on an ICACHE miss to complete [fill]
  – Forcible context switch [via setting the context switch bit]
  – Fence [equivalent to setting the context switch bit]
CHOMP Instruction Format

• CHOMP Base ISA includes one RISC format
  – Three eight-bit register operand fields
  – Opcode Field [8-bits]: defines the instruction “class”
  – Function Field [8-bits]: defines the instruction
  – 16-bit immediate field
  – 8-bit control field
  – Optional 64-bit immediate value in the next instruction word
CHOMP Instruction Format
CHOMP Instruction Format cont.

- Context switch
- Breakpoint set
- 64-bit immediate present
- Unused
- Step from breakpoint
- Register Operand 0 is a Control Register
- Register Operand 1 is a Control Register
- Register Operand 2 is a Control Register
# CHOMP Operation Classes

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Function</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Load/Store</td>
<td>Yes</td>
</tr>
<tr>
<td>0x01</td>
<td>Arith-Misc</td>
<td>Yes</td>
</tr>
<tr>
<td>0x02</td>
<td>Arith-Integer</td>
<td>Yes</td>
</tr>
<tr>
<td>0x03</td>
<td>Arith-Float</td>
<td>No</td>
</tr>
<tr>
<td>0x04</td>
<td>Arith-UDEF</td>
<td>No</td>
</tr>
<tr>
<td>0x05</td>
<td>Arith-Flow Ctrl</td>
<td>Yes</td>
</tr>
<tr>
<td>0x06</td>
<td>Arith-Atomic Full/Empty</td>
<td>Yes*</td>
</tr>
<tr>
<td>0x07</td>
<td>Arith-Thread Ctrl</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- **User-Defined Arith Operation Class**
  - Permits customer architects to define their own arithmetic instructions
  - Zero, one, two, three operand arith’s with predefined function codes
  - All user-defined arith’s obey the standard context-switch and hazard mechanisms
  - Ability to attach these user-defined instructions to user-defined performance counters

*required for the workload manager
APPLICATION EXAMPLES
#pragma cny coproc {

#pragma omp parallel for shared(p) private(tmp,j,k)

for( i=0; i<NUM_PAGES; i++)
{
    /* accumulate PR of incoming links */
    for( j=0; j<p[i].ni; j++)
    {
        k = p[i].in[j];
        tmp += ( p[k].rank/p[k].no );
    }

    /* normalize the PR */
    p[i].rank = (1-DAMP)+DAMP * tmp;
}

}
Pointer Chasing Example

- Pointer chasing
  - Multi-source graph searches
  - Multi-source shortest path searches
  - Vertex coloring
  - Some community detection algorithms

```c
//-- Parallel for N starting points
cur = *start;
for( i=0; i<iters; i++ ) {
   visited[i] = cur;
   cur = cur->next;
}
```

*benchmark represents 65K vertices per thread; vertices randomized using LCG*
Acknowledgements

• Co-authors
  – Kevin Wadleigh
  – Joe Bolding
  – Tony Brewer
  – Dean Walker

• RTL Team
  – Dean Walker
  – Mike D’Jamoos
  – John Amelio
  – Ryan Akkerman
  – Mike Dugan

• MX-100 Platform Team

• Compiler Team
  – Daniel Palermo
  – Geoff Rogers
  – Jason Eckhart
  – Randy Meyer
  – Rich Bleikamp
  – Mike Carl

Questions/Comments?
jleidel<at>conveycomputer<dot>com
THE WORLD’S FIRST HYBRID-CORE COMPUTER.